

NAVAL POSTGRADUATE SCHOOL
Monterey, California



THESIS

JAVA BASED DATA CONNECTIVITY

by

Gary D. Garingo

September 1997

Thesis Advisor:
Co-Advisor:

Luqi
V. Berzins

Approved for public release; distribution is unlimited

19980417 157

DTIC QUALITY INSPECTED 4

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302 and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE September 1997	3. REPORT TYPE AND DATES COVERED Master's Thesis	
4. TITLE AND SUBTITLE JAVA BASED DATA CONNECTIVITY		5. FUNDING NUMBERS	
6. AUTHOR Garingo, Gary D.			
7. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, CA 93943-5000		8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSOR/MONITORING AGENCY NAME(S) AND ADDRESS(ES)		10. SPONSORING/MONITORING AGENCY REPORT NUMBER	
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense of the U.S. Government.			
12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited		12b. DISTRIBUTION CODE	
13. ABSTRACT (Maximum 200 words) Current web database connectivity solutions lack an object-oriented architecture for java applications. In particular, Java is object-oriented and most legacy databases are relational. This thesis proposes a design and implementation of an object-oriented java database class hierarchy for relational database interfaces. The work reported here consists of: analysis of Java Database Connectivity specification, discussion of two-tier and three-tier architectures for database systems, mapping of relation database structure to an object model, and development of a java based framework to exercise the JDBC interfaces. This work provides (1): an object model for the relational database; (2) Integration with a middleware application for network connectivity ; (3) A Java application client to support SQL access and manipulation			
14. SUBJECT TERMS Java, Database		15. NUMBER OF PAGES 79	
		16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UL

Approved for public release; distribution is unlimited

JAVA BASED DATA CONNECTIVITY

Gary D. Garingo
B.S.E.E., California State Polytechnic University, Pomona, 1990

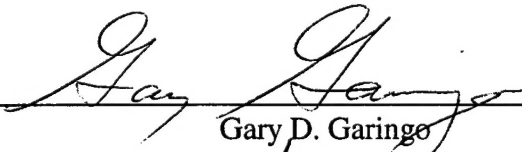
Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN SOFTWARE ENGINEERING

from the

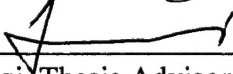
**NAVAL POSTGRADUATE SCHOOL
September 1997**

Author:

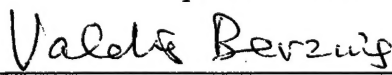


Gary D. Garingo

Approved by:



Luqi, Thesis Advisor



V. Berzins, Co-Advisor



T. Lewis, Chairman, Department of Computer Science

ABSTRACT

Current web database connectivity solutions lack an object-oriented architecture for Java applications. In particular, Java is object-oriented and most legacy databases are relational. This thesis proposes a design and implementation of an object-oriented java database class hierarchy for relational database interfaces.

The work reported here consists of: analysis of Java Database Connectivity specification, discussion of two-tier and three-tier architectures for database systems, mapping of relation database structure to an object model, and development of a java based framework to exercise the JDBC interfaces.

This work provides (1): an object model for the relational database; (2) Integration with a middleware application for network connectivity; (3) A Java application client to support SQL access and manipulation

TABLE OF CONTENTS

I.	INTRODUCTION	1
	A. GENERAL	1
	B. PROBLEM STATEMENT	2
	C. SCOPE	2
II.	BACKGROUND	5
	A. JAVA	5
	B. MSRR	5
	C. ARCHITECTURE CONSIDERATIONS	6
IIII.	PHASE 1: RESEARCH IN DATABASE SPECIFICATIONS	9
	A. RELATIONAL DATABASES	9
	B. JAVA DATABASE CONNECTIVITY SPECIFICATION	10
IV.	PHASE 2: SYSTEM CONFIGURATION	13
	A. NETWORK APPLICATION SERVER SELECTION	13
	B. WEBLOGIC T3SERVER	14
	C. NETSCAPE LIVEWIRE	14
	D. SYMANTEC DBANYWHERE	14
V.	PHASE 3: DATABASE DESIGN	17
	A. DESIGN OF DATABASE OBJECT MODEL	17
VI	PHASE 4: DESIGN OF DBJAVA MODULES	21
	A. PACKAGE DESCRIPTION	21
	B. FRAME MODULES	22
VII.	IMPLEMENTATION INSIGHTS	27
	A. SERVER-CLIENT MAINTENANCE	27
	B. SCALABILITY	27
	C. IMPEDANCE MISMATCHING	28
VIII.	FUTURE WORK	29
	A. CORBA	29
	B. OODBMS	30
	APPENDIX	31
	REFERENCES	63
	INITIAL DISTRIBUTION LIST	65

LIST OF FIGURES

Figure 1:	Client-Server Architectures	7
Figure 2:	JDBC API Components	11
Figure 3:	Architecture of Driver Interfaces	13
Figure 4:	Database Object Model.	17
Figure 5:	Graphical User Interface.	22
Figure 6:	MainFrame Connection Screen.	23
Figure 7:	MainFrame Database Tables Screen	23
Figure 8:	Browse Screen.	25
Figure 9:	Edit Frame	25
Figure 10:	MetaData Screen.	26

THESIS DISCLAIMER

Java is a registered trademark of Sun Microsystems.

ODBC, Windows NT is a registered trademark of Microsoft.

T3Server, jdbcKona is a registered trademark of WebLogic.

DbANYWHERE, Visual Café is a registered trademark of
Symantec Corp.

Netscape Enterprise is a registered trademark of Netscape
Communications.

I. INTRODUCTION

A. GENERAL

The advent of the Internet and World Wide Web (WWW) has opened opportunities for repository computing away from the mainframe to the user's desktop. Application and database servers are still needed to support the legacy infrastructure but a new class of applications presented through a web browser or web application has evolved. Web computing has fostered the proliferation of HTTP (hypertext transaction protocol) servers, common gateway interfaces (CGI), and web-crawling robots.

Java, developed by Sun Microsystems, has become a leading language for web based interfaces and is becoming an industry standard in WWW browsers and tools. Java is a simple and robust object-oriented language used to extend the current capabilities of the hypertext markup language (HTML) and the limited options in CGI programming through browser-based programs known as applets. Java also supports the development of portable applications requiring Java Virtual Machines for execution.

Government agencies are now embracing Internet and repository technologies in the execution of their information distribution model. The Modeling and Simulation Resource Repository (MSRR) is one such program. The MSRR is a collection of M&S resources and resource references implemented using a distributed system of HTTP servers connected through the World Wide Web. The current system includes pointers to disparate DoD modeling and simulation systems data, but does not provide seamless access to the underlying database management engines. The MSRR consists of a registration and catalog database which holds information on users, resources, and organizations within the M&S community. It is designed to facilitate distribution, integration, and interoperability between M&S programs and is the infrastructure for various data standardization efforts under the Defense Modeling and Simulation Office.

Portability and common interfaces are needed by the MSRR for maximum utilization and distribution to the M&S community. Database browsing and editing applications in Java would extend the capabilities of the current system and support research of issues in the design of

the next generation MSRR. A foundation for integration of future database connectivity technologies needs to be addressed.

B. PROBLEM STATEMENT

The focus of this thesis is to design and implement an object-oriented java database class hierarchy for relational database interfaces. The interfaces will support object access to relational data.

This class hierarchy will be a foundation for an object based Modeling and Simulation Resource Repository. The objective of the prototype is to exercise object access to a relational database. In future implementation, the user and resource data types in the MSRR registration model would be prime candidates for object modeling.

C. SCOPE

The class hierarchy is used to demonstrate the feasibility of an object data model for web distribution. The thesis consists of four phases. The work illustrates the use of the Java Database Connectivity (JDBC) Application Program Interface (API), two-tier and three-tier architectures, and relational database structure mapping to an object model.

The first phase involve research of the Java Database Connectivity specification. An understanding of the JDBC is needed to select appropriate technologies for a limited prototype. The JDBC API available in Java Development Kit (JDK) 1.1.3 will be used.

The second phase involves selection and integration of a Java based Network Application Server (NAS) into the overall architecture. The Application Server dynamically loads and executes modules when required by clients. In our example, the modules needing dynamic access are different JDBC drivers. The architecture supports two-tier and multi-tier connection. An ODBC (Open Database Connectivity) interface is also used in illustrating different types of connectivity. Several NAS products were examined with an appropriate one integrated into the final prototype.

An object-oriented model is used for the representation of a relational database in phase three. The MSRR registration data model was not available for the prototype implementation. Instead, a

representation of the relational database physical structure was created. The prototype demonstrates Java access to applicable data structures of one-one and one-many tables.

The object model is limited to scope due to time constraints but supports minimum requirements needed for data population under the MSRR data model. Several alternatives to the class hierarchy were found to be developing through the course of this thesis but were either infant technologies or incomplete.

The final phase involves prototyping of a java database application client. The application was demonstrated on Unix and Windows NT operating systems. It includes a schema module, data browsing module, and data editing module, and a connection module. The connection module illustrates two-tier, multi-tier and ODBC (Open Database Connectivity) operation. Other modules envisioned but outside the scope of this thesis are: CORBA (Common Object Request Broker Architecture) module and OODBMS (Object Oriented Database Management System) module.

A test site of current MSRR databases were available. Current identified systems include an Oracle database and Microsoft Access database residing on an Intel Pentium server. Microsoft's Windows NT operating system is being used due to lower software costs and availability of ODBC software.

II. BACKGROUND

A. JAVA

Java was originally a language developed to program electronic devices like VCRs and remote controls. Java is a "simple, object-oriented, network-savvy, interpreted, robust, secure, architecture neutral, portable, high-performance, multi-threaded, dynamic language"[Ref 1, p4] developed by Sun Microsystems. Java can take the form of three separate languages: BASIC-like interpretive scripting known as javascript; web browser extensions with limitations in memory, file, and database access known as applets; and a full development language without the restrictions for applications [Ref 2.].

Java is a partially compiled language. Java syntax was designed to resemble C++ to take advantage of the industry proliferation of C++ training. The Java Virtual Machine (JVM) is the core technology that makes java portable. The JVM executes byte codes which are compiled instructions at the lowest level possible without making them machine dependent. Byte codes make java, 80% compiled and 20% interpreted at run time.

Java is fully object-oriented including dynamic binding but only supports single inheritance. Pointers, references, friend functions, operator overloading, and direct memory allocation and deallocation are not allowed in Java. These restrictions are needed to make Java simple and robust.

B. MSRR

The Modeling and Simulation Resource Repository (MSRR) is an information repository shared by the modeling and simulation community. Funded by the Defense Modeling and Simulation Office (DMSO), the MSRR integrates various armed services web sites under a common registration and resource model accessible by web browsers. The system is being developed under a five-year spiral development plan currently in its second year. The MSRR is deployed on the Internet for the unclassified customers and SIPRNet (Secret IP Routing Network) for the classified community.

The requirements for the MSRR were developed to identify needs for a web based information network up to the year 2000. Support for legacy

systems, mainly relational database systems, and future integration with DMSO's HLA (High Level Architecture), distributed object architecture for future models and simulations, drive the incremental prototyping of the system.

C. ARCHITECTURE CONSIDERATIONS

In the design, the client-server architecture will be used (Figure 1). A traditional application program that wanted to access a database called its own data management libraries, or started its own invocation of the DBMS program. Disadvantages to this approach were wasted computer resources and limited support of transactions across multiple programs and users. In a client-server environment, a single multi-threaded DBMS starts up before any client program. Client requests, in the form of queries or updates, are received over the network by the DBMS, which sends back data (SQL rows or status information) in response to each request. This illustrates a two-tier architecture between the application and the database management system.

Problems with traditional two-tier architecture have become apparent. In the RDBMS market, each vendor supports a different variant of SQL. Consequently, vendors supply a different client library with a proprietary API for communicating with its servers, such as DBLIB for Sybase or OCI for Oracle. The application logic is completely different to communicate with DBMS servers from different vendors. Microsoft's Open Database Connectivity (ODBC) is an attempt to standardize the interface, but it offers a lowest common denominator approach and is seldom fully implemented. Both the DBMS-vendor client libraries and ODBC suffer from too low-level an abstraction. Writing code that deals with fetches and updates on individual records is the norm, rather than high-level operations on sets of records.

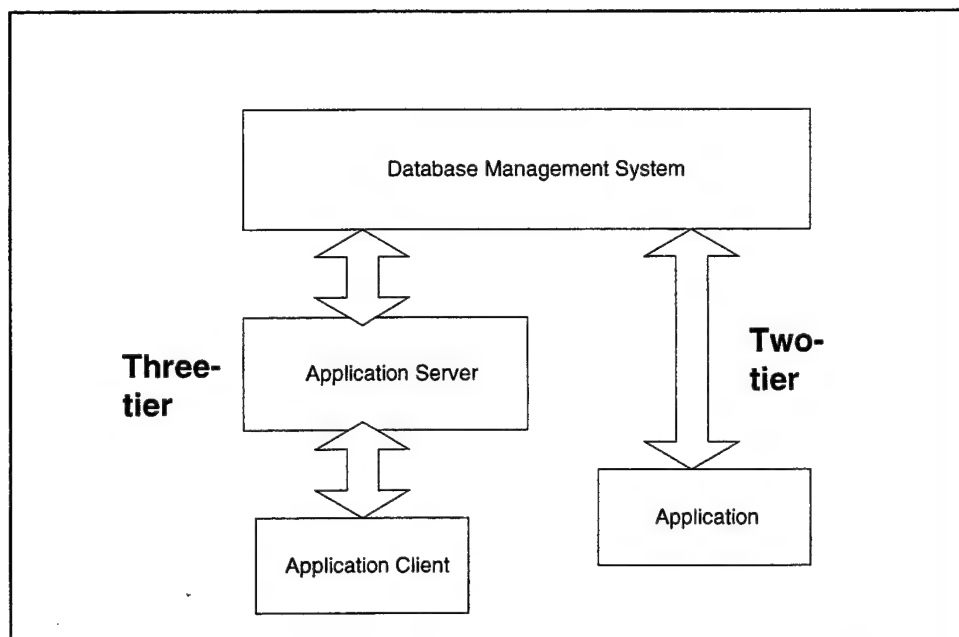


Figure 1: Client-Server Architectures

Client applications are usually deployed on desktop PCs with limited memory and disk capacity. As java and network computing is embraced, the requirements of a two-tier system cannot be satisfied by the new breed of thin-client systems. Consequently with client-server applications and the PC operating systems become larger and more complex, the cost to equip every client PC with appropriate software becomes prohibitive. In a large organization, client programs may reside on thousands of network PCs. Each time a DBMS vendor comes out with a new revision, client libraries must be updated on every PC, which is expensive as well as a system management configuration problem.

A multi-tier architecture has developed to address these problems. Multi-tier architecture (also known as three-tier or middleware) extends the standard client-server architecture (two-tier) by placing a multi-threaded application server between the client and the DBMS. Clients communicate with the DBMS through the application server, using high-level, vendor-independent requests and replies. The application server also provides the seamless network communication for software components physically distributed. The application server is responsible for executing those requests, and interfaces with the DBMS vendor's client library to communicate with databases. This third tier is used to allow

portability between different database engines. The java application is utilized as a client denoting the first tier. The second tier is occupied by a java application server which handles the passing of info between the client and the third tier, the database itself.

The prototype application will demonstrate three methods of connectivity: two-tier, three-tier, and an ODBC bridge. Besides ease of use, connectivity configuration will also be examined. A commercially available network application server and JDBC driver will be used. Development of a NAS and JDBC driver is beyond the scope of this thesis. Commercial products are rapidly being introduced to address the NAS and JDBC market.

III. PHASE 1: RESEARCH IN DATABASE SPECIFICATIONS

A. RELATIONAL DATABASES

The Relational Database Management System (RDBMS) was developed to implement the Entity-Relationship model. The ER model characterizes the information world into entities and relationships between entities. These entities or relationships could be modeled into two dimensional data structures known as tables. Tables consist of rows and columns to map a particular data element. Columns can be designated as keys, primary or foreign, which are constrained to be unique values for every row occurrence. A primary key denotes a column, or set of columns, where its data value allows a particular row to be unique among the other rows in its table. A foreign key consists of a column, or set of columns, which map a particular row to another row in a related table.

SQL is the non-procedural language used to access data within a RDBMS. SQL is an evolving standard adopted by various standards bodies with most major vendors supporting some flavor (often SQL92) of the language. SQL is blocked into transactions and does not flow with normal program structures. This problem, known as impedance mismatch, is partially solved through the use of cursors which transverse data sets by retrieving a row at a time.

SQL can be classified into the following types: DQL (data query language), DDL (data definition language) and DML (data manipulation language). The DQL encompasses the commands and terms need to access the data; the DDL commands allow for creation of the entities; consequently DML commands are used to edit existing entities.

Language designers needed to describe set manipulations for relational data. The choice of SQL was to make each statement stateless and to embed the code. SQL was developed as a means to express relational calculus in a fair compromise between the needs of traditional third-generation language programmers and set theory [Ref. 3, p. 22]. Embedded SQL within program statements, iscompiled through a preprocessor and translated into functions.

The following command constructs are supported in this java prototype:

SQL Type	Commands
DQL	select
DDL	insert
DML	update, delete

Table 1. SQL Commands

ODBC was a mechanism for Microsoft to force RDBMS vendors to talk the same language. It was a first step in providing portable application integration but proved to be slow and unwieldy. Poor syntax, distorted error messages, and difficult installations were the norm. Future revisions have somewhat improved the performance of ODBC interfaces but client installation still was not very scalable in the enterprise environment.

B. JAVA DATABASE CONNECTIVITY SPECIFICATION

JDBC is based on the xOpen standard of ODBC. ANSI SQL92 is required for full JDBC compliance. JDBC is an extension to the java language under the java.sql package and has become standard in release 1.1 of Java. Two interfaces are available in JDBC: JDBC API and JDBC Driver API. Most database application programmers will use the JDBC API while most RDBMS vendors will use the JDBC Driver API for native libraries.

The JDBC API is known as a 'call level' API where it supports the passing of raw SQL statements to the underlying DBMS driver [Ref. 4]. The classes that are most dealt with for database interfaces are: java.sql.Connection, java.sql.Statement, java.sql.ResultSet, and java.sql.DriverManager. The interface registers with the DriverManager which keeps track of various Connections to external databases. A DriverManager allows loading of a particular driver; a Connection represents a connection to a particular instance of a database using a selected driver; a Statement contains the executing SQL statement for a given connection; and a ResultSet provides access to the row results for a given statement. Figure 2 provides a description.

Databases are identified by a JDBC Uniform Resource Locator i.e. jdbc:<sub-protocol>:<DBMS specific identifier>. There are also security

limitations for applets using JDBC: the Java Security Manager requires untrusted applets to only access the sources from the server from which it was loaded. This limitation resulted in a decision to implement the prototype client as an application rather than applet.

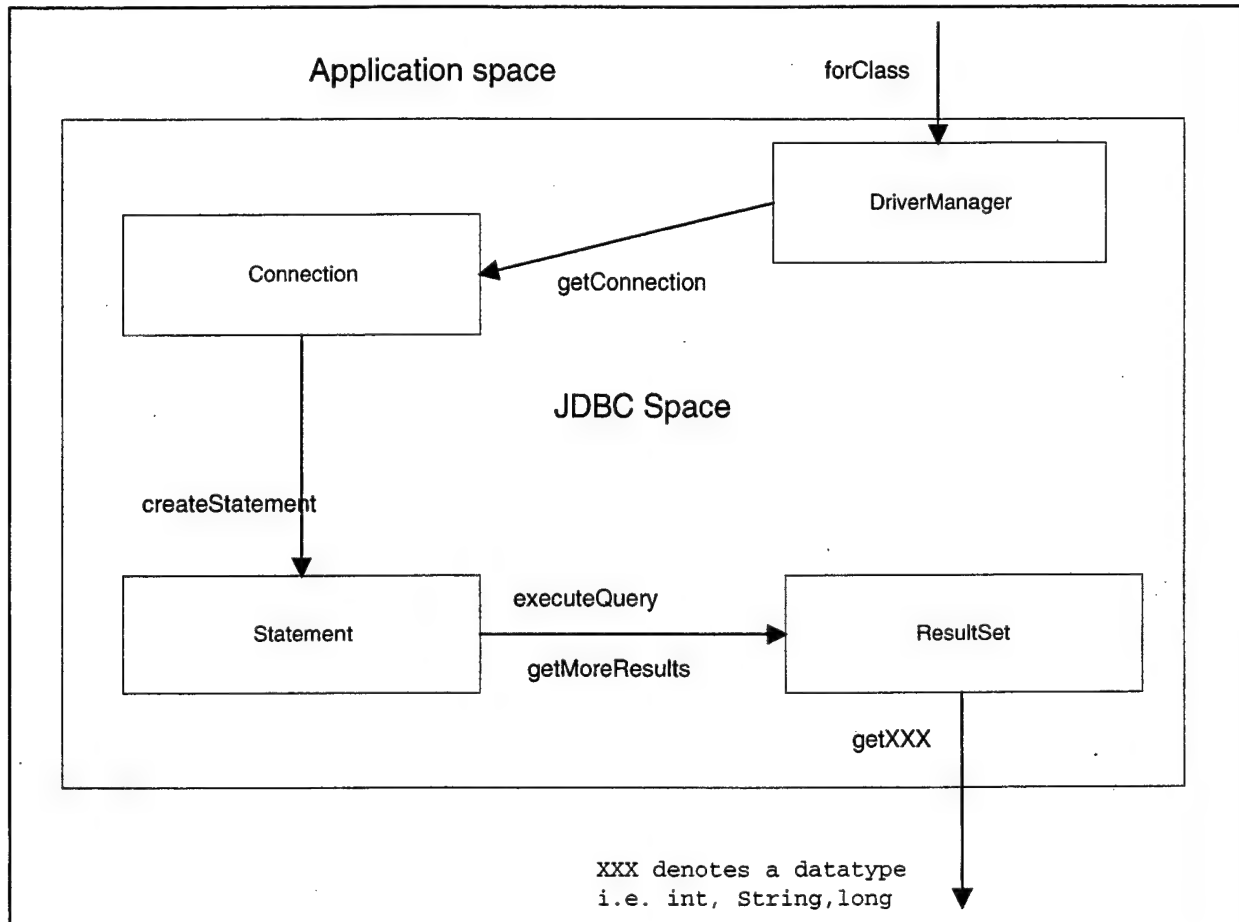


Figure 2. JDBC API Components

The JDBC Driver API encompasses the above classes but as native interfaces to the underlying DBMS. This requires java bridges to C libraries since many major DBMS vendors do not yet support java client software. Non-DBMS vendors have developed various drivers for many of the major RDBMS engines. An obvious drawback is the non-portability of the native bridge solution compared to completely pure java.

No matter what JDBC driver is being used, the normal steps needed to interface with the database are:

1. Import java.sql package
2. Load a JDBC Driver using the DriverManager
3. Open a connection to the database by providing at least JDBC URL, account name and password

4. Create and execute the SQL Statement
5. Retrieve the ResultSets or SQL Error Code from the connection
6. Decompose and translate the data values into appropriate Java data types
7. Close the connection

Program code examples are included in the Appendix under the MainFrame.java functions.

IV. PHASE 2: SYSTEM CONFIGURATION

A. NETWORK APPLICATION SERVER SELECTION

A NAS needed to be selected for three-tier connections. Rather than develop one from scratch, an appropriate commercial off-the-shelf (COTS) product was procured. The MSRR project requires a "pure" java implementation and platform compatibility for Sun, Hewlett-Packard, and Windows NT.

For the java prototype, an Oracle JDBC driver developed by WebLogic and the JDBC-ODBC Bridge developed by Sun Microsystems are used. The WebLogic network application framework supports both two-tier and three-tier connections while the JDBC-ODBC implements a two-tier connection. The burden of the network and driver management would fall to the NAS. A suitable design is illustrated in Figure 3.

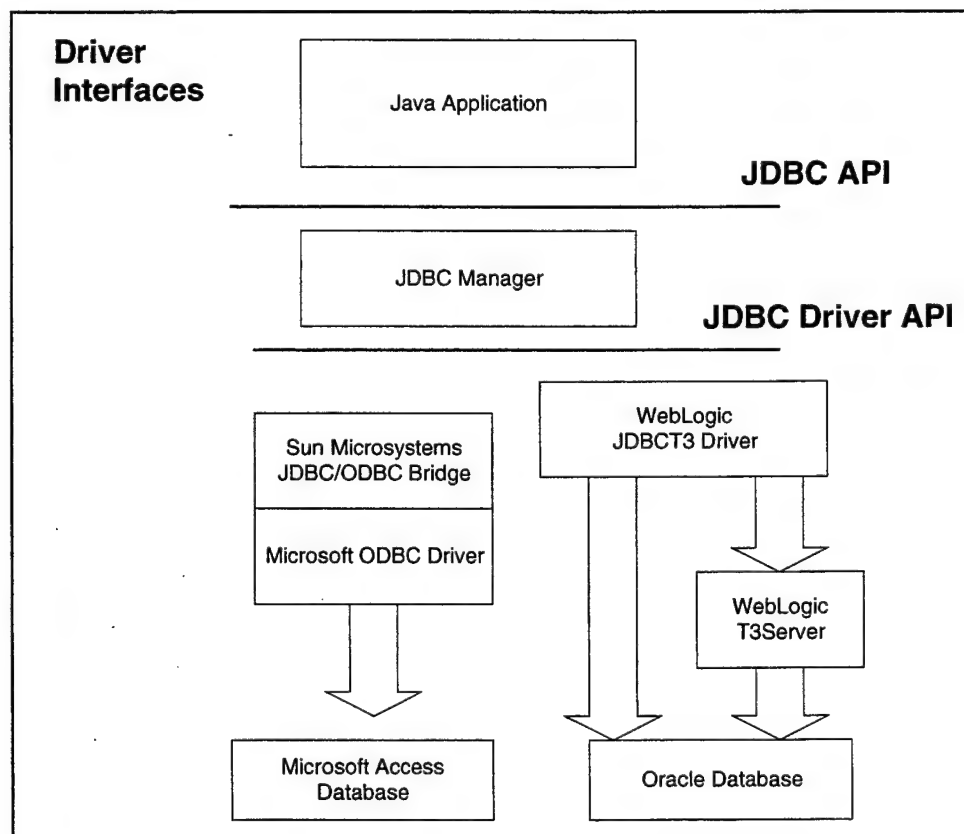


Figure 3. Architecture of Driver Interfaces

B. WEBLOGIC T3SERVER

WebLogic has developed a multi-tier java application framework for client-server computing. Its T3Server environment provides solutions for database connectivity, event management, and remote computing. Since the T3Server is completely written in java, the product provides straightforward interfaces to the JDBC API. WebLogic also provides a JDBC driver suite (jdbcT3) which supports both two-tier and three-tier implementations. For this prototype, a T3Server with jdbcT3 was procured to support Oracle and Microsoft Access.

C. NETSCAPE LIVEWIRE

Another alternative to JDBC that was examined but not selected is Netscape's LiveWire application suite. LiveWire is a client-server architecture that extends the web server interface. Because Netscape developed LiveWire, it can only extend its suite tools, Enterprise Server resulting in non-portability of developed code. The code supported by the LiveWire server application is javascript through a three-tier client-server architecture. Clients are web browsers that can handle application logic like data validation. The servers are the relational database management systems, which house the data, metadata, and enforce referential integrity. The web HTTP server with the LiveWire extension performs the middle tier or middleware application. This extension resembles the common gateway interface client but uses the javascript parameter passing rather than an html form parameter. Given LiveWire's dependence on javascript, the product was not used in the prototype.

D. SYMANTEC DBANYWHERE

Symantec's dbANYWHERE Workgroup Server provides three-tier client/server database connectivity. It is thoroughly integrated with Symantec's Visual Café Pro, a java graphical user interface builder, for its java connectivity. DbANYWHERE supports design time database interaction and facilitates the generation of database aware (dbAWARE) Java forms that communicate with databases through the dbANYWHERE server. DbANYWHERE can access Oracle, Sybase, MS SQL Server, Sybase SQL Anywhere, and MS Access databases, and includes an ODBC link that provides connectivity to over 30 other databases with the addition of an ODBC driver.

One of the shortcomings of the product though was the integration of Café Pro. Café Pro offers another toolkit that extends the java AWT (Abstract Window Toolkit) API. Visual Café utilizes GUI components known as DataWizards. The DataWizards automatically generates java code to perform SQL commands on a selected database and its tables. For example, DataWizard buttons for fetching, updating, and deleting are made available to your application by dragging the component onto your program's window frame. The java code generated is DbANYWHERE API calls which in turn call the JDBC API. Since one of the objectives is to exercise straight JDBC calls, the DataWizard code did not fulfill that requirement. Separating DbANYWHERE and Visual Café Pro to exercise the thesis objectives was not possible as the generated code referred to database functions that were not documented in the software help package.

V. PHASE 3: DATABASE DESIGN

A. DESIGN OF DATABASE OBJECT MODEL

In order to map a relational data model to an object oriented language, an object model was devised. The representation of the relational model as objects allows java structures to facilitate data exchange between the database management system and the java application. The object model at a minimum needs to support schemas, tables, and columns. Other entities like indexes, views, and synonyms are modeled but not fully implemented.

An object representation of the table structure (Figure 4) is useful in developing graphical user interfaces to the data. The most useful entities within the relational model are the schema, table, and column. A particular database can hold several schemas; a schema can hold many tables; and a table can contain several columns.

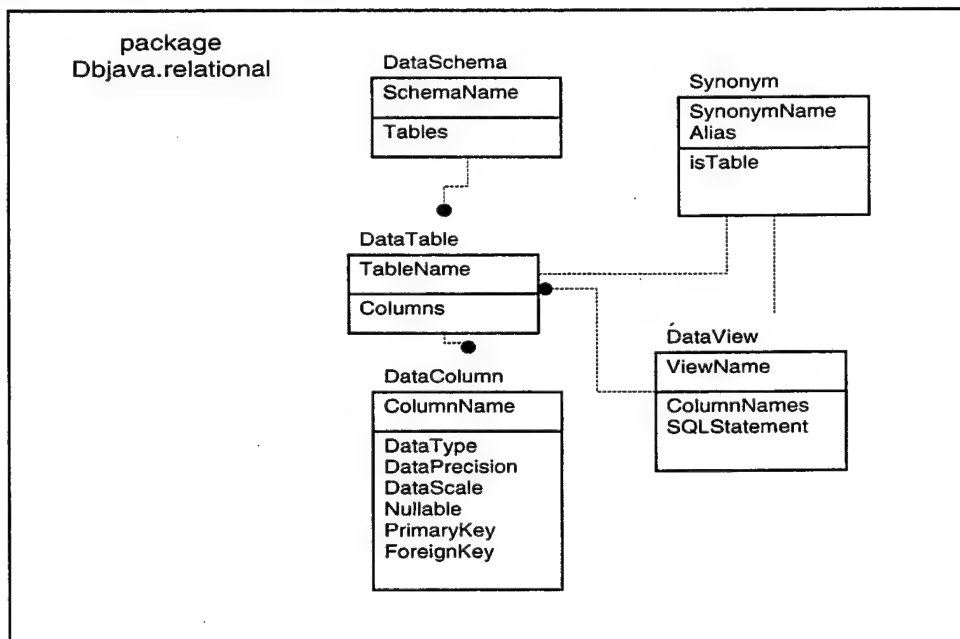


Figure 4. Database Object Model

Impedance mismatching makes object modeling of a relational database a problem. An object with its attributes, associations, and behaviors does not fit nicely into a flat row and table structure [Ref. 5.]. There is also the issue of storing and retrieving objects. The simplest and most direct mapping from a relational to an object schema is the one-to-one mapping of a table to a type, and the mapping of

columns in a table to properties in a type known as TTM (Table-to-Type Mapping) as discussed in Ref. 6. The mapping normally results in single types but also can take the form of multiple types like different columns of the same table map to two non-inheriting types. The four TTM approaches are:

1. table-to-type
2. table-to-multiple-types
3. table-to-inherited-types
4. multiple-tables-to-inherited-types (type-tree)

Since this was a prototype for feasibility, the simple table-to-type mapping was used throughout the model. An exercise in table-to-multiple-types would be necessary to support the MSRR registration data model.

Typically, a type will include mappings for all of the columns; however optimally there would be no requirement that all columns in a table be mapped to properties. Then the type could be defined to include only mappings to properties for those columns required by applications. Mapping the primary key is not a requirement especially when the key provides no extra value besides providing a unique identifier for the tuple. This rule provides a degree of flexibility in the model. [Ref 6, p. 72]

The relational classes are developed under the DbJava.relational package. The relational classes will be used to contain the data retrieved from the RDBMS into an object that can easily be manipulated by java operations. There is a semantic mismatch between the object syntax and procedural constructs of the relational model. One of the benefits of object syntax was ease of maintenance and reduced complexity. Integrating a relation model for data persistence contradicts these two benefits of object programming.

The relational model delivers data in the form of two-dimensional matrixes (column and row). JDBC implements the matrix in the form of a java.sql.ResultSet. In order to access the data, two cursors are needed to transverse the matrix. By specifying a row and column cursor, the data value can be retrieved then translated to an appropriate object type needed for a particular class. Some object instances may require

its data to be stored in separate rows in different tables. One initialization statement in java can result in 20 or more JDBC execution statements to retrieve data to populate the object. This interface introduces more complexity but can not be avoided.

Most information needed to describe a data value resides in the `DbJava.relational.DataColumn` class. The `DataColumn` describes the data type (string or number or date), precision (if number, digits after the decimal point), and scale (size of number or length of characters). Other attributes available are whether the column could be null (nullable) and if the column is a primary or foreign key. Primary and foreign keys could have also been modeled as sub-classes of the `DataColumn` class, but they were relegated to attributes to simplify the class hierarchy.

Vector classes (dynamic array data structures) were used to represent the one-to-many relationships between schemata to tables and tables to columns. Vectors provide convenient accessor and destroyer functions to simplify coding. Though vectors provide built-in behavior for managing the array, it still requires multiple loops and casting calls to finally retrieve data. This cumbersome technique was a disadvantage in using JDBC. Oracle has responded to this shortfall by proposing a standard known as J/SQL. J/SQL will be discussed in a later section.

The `DataRecordTextField` class was needed to translate non-string or non-numeric data types into the DBMS data types. Here is where the legacy of dynamic SQL is still apparent. JDBC driver transforms the data value into appropriate java data types, but the programmer still has the ability to convert between different java types like a Integer type can be retrieved as a String type. This was appropriate for the date data type as `java.Date` class did not match the Oracle date type. For future development, the construction of distinct GUI collections or objects that support the editing and display of each unique data type would be more useful i.e. MSRR requirement. Examples are a date spinner, combo boxes and check boxes for enumerated data types, and radio button groups for a boolean data type.

VI. PHASE 4: DESIGN OF DBJAVA MODULES

A. PACKAGE DESCRIPTION

The prototype client application is not discussed in detail as the program code is provided in the Appendix. A general discussion follows.

The package DbJava contains the code for the java application client. DbJava.relational module holds the description of the database entities. From the passing of a DataSchema object to the BrowseFrame, EditFrame, MetaDataFrame, or SchemaFrame, the frame is populated and configured per the database entity description. Though different GUI objects are used in the various frames, one single object (DataSchema) is the target for all manipulations.

This design in the application followed the MVC (model-view controller) design pattern. The model was provided by the DbJava.relation package; the view was developed in all the frame objects; while the controller was implemented in the frame operations. MVC is a design pattern first used in the Smalltalk-80 programming language to separate data, GUI, and application logic.

The design philosophy is to present data in different views configured by independent controllers. MVC decouples views and models by establishing a subscribe/notify protocol between them. A view must ensure that its appearance reflects the state of the model. Whenever the model's data changes, the model notifies views that depend on it. In response, each view gets an opportunity to update itself. This approach let you attach multiple views to a model to provide different presentations [Ref. 7]. Figure 5 provides the class GUI.

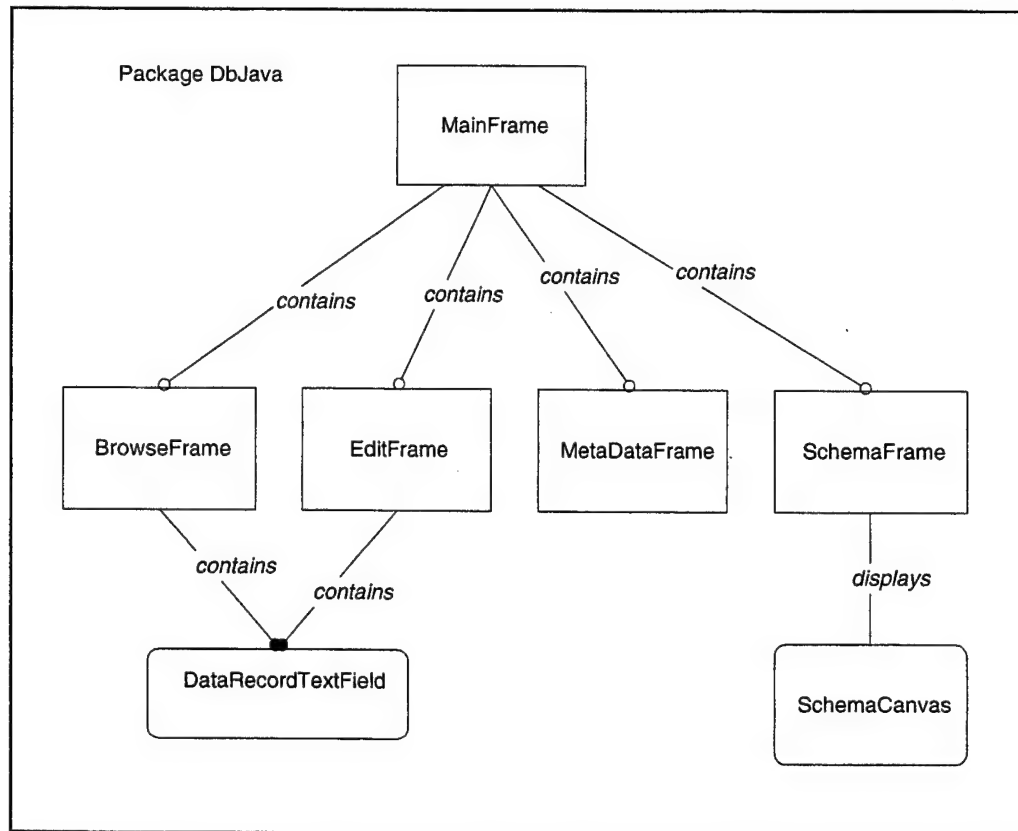


Figure 5. Graphical User Interface

B. FRAME MODULES

The MainFrame (Figure 6) is where user and database information is specified. This is also the connection module. With user information, the client can connect to the T3Server through a three-tier, two-tier, and ODBC bridge interface. A message box is also provided for error information, normally networking, driver configuration, and database availability. After connection, the MainFrame pages to the main panel (Figure 7). This panel allows selection of a table in the connected database to be accessed in different modes: browse, edit, meta-data, and schema. A disconnect mode is also provided to allow the user to exit and clear all current connections.

DbJAVA ©C. D. Garingo, 1997

Login Name:

Password:

T3Server:

Database:

Please log in.

Figure 6: MainFrame Connection Screen

DbJAVA ©C. D. Garingo, 1997

Available Tables:

BASLINE
BASLINE_REQ
HISTORICAL
ODOMETER
ORGANIZATION
REQUIREMENT
REQUIREMENT_SOURCE
STATUS

Table Name:

Figure 7: MainFrame Database Tables Screen

The BrowseFrame (Figure 8) allows browsing of all rows in a table. Popups are created which display all unique entries for each column. A fetch button selects all rows within the selected table by retrieving a DataSet of the table. A next button displays each individual row in the DataSet through the textfields.

The EditFrame (Figure 9) allows editing of particular rows in a table. Insertion of new rows is also allowed in this mode along with updating and deleting rows.

The MetadataFrame (Figure 10) was designed to take advantage of metadata operations provided by JDBC. JDBC implements a java.sql.MetaData class but vendor support has been inconsistent at this point of time. Oracle did not support the catalog object and its accompanying function calls in java.

The SchemaFrame module constructs a graphical representation of a selected database schema. This module would be useful as a documentation or reverse-engineering tool. It was not fully implemented because of time constraints but the basic skeleton needed for component drawing was developed through the SchemaCanvas class. With a SchemaCanvas object, each individual drawing of a table represented an instance of this class. This specification offered the flexibility to extend the class for added functionality without affecting the SchemaFrame object itself.

Table Browser

fetch next

REQUIREMENT_ID		NULL
REQ_SOURCE_ID		NULL
DOC_REQ_ID		NULL
REQ_DESC_SHORT		NULL
REQ_DESC_LONG		
REQ_SOURCE_PARA_TITLE		NULL
REQ_SOURCE_PARA_NUMBER		NULL
REQ_DATE_SUBMITTED		NULL
REQ_PRIORITY		NULL

Figure 8: Browse Screen

Table Editor

fetch next new record insert update delete

REQUIREMENT_ID	
REQ_SOURCE_ID	
DOC_REQ_ID	
REQ_DESC_SHORT	
REQ_DESC_LONG	
REQ_SOURCE_PARA_TITLE	
REQ_SOURCE_PARA_NUMBER	
REQ_DATE_SUBMITTED	
REQ_PRIORITY	

Figure 9: Edit Frame

Untitled

Table: **BASELINE**

Show

☐ Primary Key ☐ Data Types

☐ Foreign Keys ☐ Data Constraints

BASELINE			
Name	Type	Nullable	Length
BASELINE_ID	NUMBER	N	22.0
SYSTEM_NAME	VARCHAR2	N	50.0
SYSTEM_ACRONYM	VARCHAR2	N	12.0
BASELINE_RELEASE_TITLE	VARCHAR2	N	30.0
BASELINE_REQ_FREEZE_DATE	DATE	Y	7.0
SYSTEM_RELEASE_DATE	DATE	Y	7.0

Figure 10: MetaData Screen

VII. IMPLEMENTATION INSIGHTS

A. SERVER - CLIENT MAINTENANCE

The two-tiered architecture required client libraries on client and server machines. Even though the JDBC driver was installed on the client machine, a network protocol driver for the database, like SQLNet for Oracle, was also required. Microsoft Access did not support this architecture as no JDBC drivers existed for it. Access was dependent on the ODBC Bridge because of its market target as a personal computer tool rather than enterprise solution.

When the client application and database were on the same machine, no connection errors were witnessed. A local connection was the preferred installation since the actual DBMS libraries were available to the java application if available on the user's environment path. This situation is strictly limiting in the remote and distributed computing configuration but was ideal for single system installations.

The ODBC Bridge was the least effective of all connections. Except for the Microsoft Access interface, ODBC suffered inconsistent connection performance and was very particular about the ODBC configuration of the underlying operating system. Sun Microsystems provides the bridge with caveats for application use and strongly recommends using a pure JDBC driver for enterprise systems. The ODBC Bridge is recommended for use in applet development and JDK 1.0.2 and JDK 1.1.1.

B. SCALABILITY

The three-tier architecture offered the best scalability due to connection management. A two-tier model forces the database server to have as many concurrent connections as possible. Every client application that connects to the DBMS must maintain a persistent state in order to support SQL transactions. With the three-tier model, the application server maintains one connection to the database but manages transactions through multi-threaded processes. It is unlikely that a NAS will be as large and cumbersome as a RDBMS.

C. IMPEDANCE MISMATCHING

The mismatch of an object and relational object resulted in large and cumbersome code for data retrieval. Oracle has proposed the use of a preprocessor for embedded SQL in Java. The preprocessor detects type mismatch and provides some bookkeeping functions. The design proposal is too new for examination but Oracle predicts 10 lines of J/SQL code replacement for 50 to 100 lines of JDBC code. [Ref 3, p.24]

VIII. FUTURE WORK

For future development, the MSRR is required to support the following object technologies: CORBA and an OODBMS. Both will have strong support in development of Java solutions because of the object nature of their technology.

A. CORBA

CORBA (Common Object Request Broker Architecture) is an object middleware architecture provided by the Object Management Group (OMG), a consortium of 700+ companies. CORBA allows intelligent components to discover each other and inter-operate over an object bus. The bus, known as an ORB (Object Request Broker) allows an extensive set of services like creating and deleting objects, accessing other objects, persistent storage, and defining ad-hoc relationships [Ref. 8, p 7-14]. For more details on CORBA, the OMG Publication, "Essential CORBA" by Thomas Mowbray and Ron Zahavi is recommended [Ref. 9].

A component's boundaries are defined through an interface specification known as IDL (Interface Definition Language). IDL is portable across languages, tools, operating systems, and networks. IDL is declarative and provides no implementation details. CORBA 1.1 specified IDL, language bindings, and an ORB API. CORBA 2.0 introduces interoperability between vendor ORBs (IIOP, Internet Inter-ORB Protocol).

The MSRR system would take advantage of IIOP in integration for the forthcoming object based models and simulations of HLA. Currently, java based ORBs are available from Visigenic and will be included in upcoming versions of Netscape's web browser. This ORB is a prime candidate to replace the Weblogic T3Server if relational database support is provided. If relational support is lacking, an application layer before the ORB and JDBC will need to be developed. Extension of a proxy server to preprocess messages between HTTP and IIOP requests is a viable solution. Several servers like Sun Microsystems Java Server and Netscape's Enterprise 3.0 suite are following that approach.

B. OODBMS

An object database can be integrated into the MSRR if the CORBA design is utilized since most OODBMS are CORBA compliant. OODBMS vendors will also support two-tier connections to their products through bindings with OO languages like C++ and java.

The Object Database Management Group (ODMG) has introduced ODMG 2.0, which describes interactions between java applications, or applets and object oriented databases. The ODMG Binding for Java specification provides a standard API to automatically store java objects in object compliant databases. It promotes portability across multiple database products and is a standard for object-relational mapping. This solution is projected to be finalized during first quarter of 1998 and would have been more useful than the java database class hierarchy.

APPENDIX

```

/* File: MainFrame.java */
package DbJava;
/*****
import java.sql.*;
import java.lang.*;
import java.util.*;
import java.awt.*;
import weblogic.common.*;
import sun.jdbc.odbc.*;
*****/

public class MainFrame extends Frame {
    // Applet globals
    String user_name;
    String user_password;
    String current_status_bar;
    boolean good_connection;
    T3Client t3;
    Driver drv;

    // Database Objects
    DatabaseMetaData dma;

    Connection conn;
    String driver_URL;
    String driver_class;
    String DB;
    String server_URL;
    String tableName;

    static public void main(String argv[]) {
        MainFrame dbb = new MainFrame();
        dbb.setVisible(true);
    }
/*****/

    public MainFrame(String title) {
        this();
        setTitle(title);
    }
/*****/

    public synchronized void setVisible(Boolean b) {
        setLocation(50, 50);
        super.setVisible(true);
    }
/*****/

    public MainFrame() {
        dblFrame = null;
        schemaPad = null;
        metaDataFrame = null;

        this.setTitle("DbJava @G. D. Garingo, 1997");
        // {{ INIT_CONTROLS
        setLayout(null);
        addNotify();
        setSize(445, 400);
        mainCardLayout = new CardLayout(0, 20);
        mainPanel = new java.awt.Panel();
        mainPanel.setLayout(mainCardLayout);
        mainPanel.setBounds(0, 432, 400);
        add(mainPanel);

        panel1 = new java.awt.Panel();
        panel1.setLayout(null);
        panel1.setBounds(0, 432, 400);
        mainPanel.add("panel1", panel1);

        panel2 = new java.awt.Panel();
        panel2.setLayout(null);
        panel2.setBounds(0, 432, 400);
        mainPanel.add("panel2", panel2);

        textStatus = new java.awt.TextArea(" ", 3, 20,
            "TextArea.SCROLLBARS_VERTICAL_ONLY");
        textStatus.setEditable(false);
        textStatus.setBounds(18,
            getInsets().height-100-getInsets().bottom-
            410,
            90);
        // textStatus.setFont(new Font("Dialog", Font.PLAIN, 10));
        panel1.add(textStatus);

        labelName = new java.awt.Label("Login Name:");
        labelName.setBounds(108, 40, 90, 18);
        labelName.setFont(new Font("Dialog", Font.BOLD, 12));

```

```

panel1.add(labelName);

text_loginName = new java.awt.TextField();
text_loginName.setBounds(222,40,120,30);
text_loginName.setBackground(Color.white);
//text_loginName.setFont(new Font("Helvetica", Font.PLAIN,
10));

panel1.add(text_loginName);

labelPassword = new java.awt.Label("Password:");
labelPassword.setBounds(108,80,90,18);
labelPassword.setFont(new Font("Dialog", Font.BOLD, 12));
panel1.add(labelPassword);

textPassword = new java.awt.TextField();
textPassword.setEchoChar('*');
textPassword.setBounds(222,80,90,30);
textPassword.setBackground(Color.white);
panel1.add(textPassword);
//just for debugging
text_loginName.setText("system");
textPassword.setText("manager");

labelI3 = new java.awt.Label("I3Server:");
labelI3.setBounds(108,120,90,18);
labelI3.setFont(new Font("Dialog", Font.BOLD, 12));
panel1.add(labelI3);

textI3 = new java.awt.Choice();
panel1.add(textI3);
textI3.addItem("dragonbreath.nosc.mil");
textI3.setBounds(222,120,120,24);

labelInstance = new java.awt.Label("Database:");
labelInstance.setBounds(108,162,108,18);
labelInstance.setFont(new Font("Dialog", Font.BOLD, 12));
panel1.add(labelInstance);

textDB = new java.awt.Choice();
panel1.add(textDB);
textDB.addItem("RQM1");
textDB.setBounds(222,162,90,24);

threeTier_button = new java.awt.Button("Three Tier");
threeTier_button.setBounds(100,210,88,23);
panel1.add(threeTier_button);

twoTier_button = new java.awt.Button("Two Tier");
twoTier_button.setBounds(200,210,88,23);
panel1.add(twoTier_button);

odbc_button = new java.awt.Button("ODBC");
odbc_button.setBounds(300,210,88,23);
panel1.add(odbc_button);

logoff_button = new java.awt.Button("Log Off");
logoff_button.setBounds(20,250,88,23);
panel2.add(logoff_button);
logoff_button.setDisabled(false);

schema_button = new java.awt.Button("Schema");
schema_button.setBounds(200,250,88,23);
panel2.add(schema_button);
schema_button.setDisabled(true);

browse_button = new java.awt.Button("Browse");
browse_button.setBounds(20,280,88,23);
panel2.add(browse_button);
browse_button.setDisabled(true);

edit_button = new java.awt.Button("Edit");
edit_button.setBounds(200,280,88,23);
panel2.add(edit_button);
edit_button.setDisabled(true);

metadata_button = new java.awt.Button("Meta Data");
metadata_button.setBounds(20,310,88,23);
panel2.add(metadata_button);
metadata_button.setDisabled(true);

message("Please log in.");
text_loginName.requestFocus();
mainCardLayout.show(mainPanel, "panel1");

// Card 2
m = new java.awt.List(0, false);
panel2.add(m);
m.setBounds(24,36,320,136);
m.setBackground(Color.white);

```

```

    }
    else if (event.target == logoff_button && event.id ==
Event.ACTION_EVENT) {
        logoff_button_Clicked(event); // user clicked Log Off
        button
        return true;
    }
    else if (event.target == schema_button && event.id ==
Event.ACTION_EVENT) {
        schema_button_Clicked(event); // user clicked Log
Off button
        return true;
    }
    else if (event.target == m && event.id ==
Event.ACTION_EVENT) {
        table_list_Clicked(event);
        return true;
    }
    else if (event.target == browse_button && event.id ==
Event.ACTION_EVENT) {
        table_list_Clicked(event);
        return true;
    }
    else if (event.target == edit_button && event.id ==
Event.ACTION_EVENT) {
        edit_button_Clicked(event);
        return true;
    }
    else if (event.target == metadata_button && event.id ==
Event.ACTION_EVENT) {
        metadata_button_Clicked(event);
        return true;
    }
    else if ((event.id == Event.KEY_PRESS) && (event.key == 9) ||
(event.key == 10)) {
        if (event.target == text_loginName) { textPassword.requestFocus(); }
        else if (event.target == textPassword) { textI3.requestFocus(); }
        else if (event.target == textI3) { textDB.requestFocus(); }
        else if (event.target == textDB) {
            threeIier_button.requestFocus();
        }
        else if (event.target == threeIier_button) {
            logoff_button.requestFocus();
        }
        else if (event.target == logoff_button) {
            text_loginName.requestFocus();
        }
        return true;
    }
    else if (event.id == Event.WINDOW_DESTROY) {
        setVisible(false); // hide the frame
        dispose();
        System.exit(0);
    }

```

```

label1 = new java.awt.Label("Table Name:");
label1.setBounds(24,204,100,24);
panel2.add(label1);

textField = new java.awt.TextField();
textField.setBounds(132,204,228,30);
textField.setFont(new Font("Helvetica",Font.PLAIN,12));
textField.setBackground(Color.white);
panel2.add(textField);

table_list_label = new java.awt.Label("Available Tables:");
table_list_label.setBounds(24,12,180,24);
panel2.add(table_list_label);
//}
//*****
//needed to override so can automatically shut down t3 client connection
when
// exiting from the applet
public void destroy() {
    close_server();
}
//*****
// handleEvent()
public boolean handleEvent(Event event) {
    if (event.target == threeIier_button && event.id ==
Event.ACTION_EVENT) {
        threeIier_button_Clicked(event); // user clicked Log In
        button
        return true;
    }
    else if (event.target == twoIier_button && event.id ==
Event.ACTION_EVENT) {
        twoIier_button_Clicked(event); // user clicked Log In
        button
        return true;
    }
    else if (event.target == odbbc_button && event.id ==
Event.ACTION_EVENT) {
        odbbc_button_Clicked(event); // user clicked Log In
        button
        return true;
    }
}

```

```

        "from cols where table_name = " + tableName + """);
        System.out.println("sql: " + sqlStmt);
        stmt1.executeQuery(sqlStmt);
        ResultSet rs = stmt1.getResultSet();

        EditFrame dbEdit = new EditFrame();
        dbEdit.addFields(rs, conn, tableName);
        dbEdit.setVisible(true);

        rs.close();
        stmt1.close();

    } catch (Exception e) {
        System.out.println("Exception occurred running query" + e);
    }
}

/*****
// user clicked the schema button
void schema_button_Clicked(IEvent event) {
    schemaPad = new SchemaFrame(conn);
    schemaPad.setLocation(new Point(100,100));
    schemaPad.setSize(400,400);
    schemaPad.show();
}
/*****/

/*****/
// user clicked the metadata button
void metadata_button_Clicked(IEvent event) {
    metaDataFrame = new MetaDataFrame(conn);
    metaDataFrame.setLocation(new Point(500,500));
    metaDataFrame.show();
}

/*****/
void threeTier_button_Clicked(IEvent event) {
    if (textLoginName.getText().length() != 0)
        user_name = textLoginName.getText();
    else {
        message("Please enter your Login Name.");
        return;
    }
    if (textPassword.getText().length() != 0)
        user_password = textPassword.getText();
    else {

```

```

        return true;
    }
    return super.handleEvent(event);
}

/*****/
void table_list_Clicked(IEvent event) {
    tableName = m.getSelectedItem(); //Set which table working with
    tableNameField.setText(tableName);
    try { //perform query

        Statement stmt1 = conn.createStatement();
        String sqlStmt = (
            "select
            column_name,data_type,data_length,data_precision,data_scale " +
            "from cols where table_name = " + tableName + """);
        System.out.println("sql: " + sqlStmt);
        stmt1.executeQuery(sqlStmt);
        ResultSet rs = stmt1.getResultSet();

        dbFrame = new BrowseFrame();
        dbFrame.addFields(rs, conn, tableName);

        rs.close();
        stmt1.close();

    } catch (Exception e) {
        System.out.println("Exception occurred running query" + e);
    }

    dbFrame.setVisible(true);
}

/*****/
void edit_button_Clicked(IEvent event) {
    tableName = m.getSelectedItem(); //Set which table working with
    tableNameField.setText(tableName);
    try { //perform query

        Statement stmt1 = conn.createStatement();
        String sqlStmt = (
            "select column_name,data_type,data_length,data_precision,data_scale "

```



```

message("Please enter your Password.");
return;
}
if (textT3.getSelectedText().length() != 0)
server_URL = "t3://" + textT3.getSelectedText() + ":7001";
else {
message("Please enter the T3Server name, e.g. dragonbreath");
return;
}
if (textDB.getSelectedText().length() != 0)
DB = textDB.getSelectedText();
else {
message("Please enter the Oracle Instance (database name).");
return;
}
threeTier_connect();
showTables();
}
/*****
void odbc_button_Clicked(Event event) {
if (textLoginName.getText().length() != 0)
user_name = textLoginName.getText();
else {
message("Please enter your Login Name.");
return;
}
if (textPassword.getText().length() != 0)
user_password = textPassword.getText();
else {
message("Please enter your Password.");
return;
}
if (textT3.getSelectedText().length() != 0)
server_URL = "t3://" + textT3.getSelectedText() + ":7001";
else {
message("Please enter the T3Server name, e.g. dragonbreath");
return;
}
if (textDB.getSelectedText().length() != 0)
DB = textDB.getSelectedText();
else {
message("Please enter the Oracle Instance (database name).");
return;
}
odbc_connect();
showTables();
}

```

```

}
/*****
public void showTables() {
// Retrieve info for panel2
if (good_connection) {
try {
Statement stmt = conn.createStatement();
stmt.executeQuery("select tname from tab where
tabtype = 'TABLE' and tname not like 'DEFI-%'");
ResultSet rs = stmt.getResultSet();

while (rs.next()) {
m.addItem(rs.getString(1));
}

mainCardLayout.show(mainPanel, "panel2");

} catch (Exception e) {
message("Exception: " + e);
}
}
}
/*****
// user clicked the logoff button
void logoff_button_Clicked(Event event) {
m.removeAll();
tableNameField.setText("");
if (dbFrame != null)
dbFrame.dispose();
if (schemaPad != null)
schemaPad.dispose();
if (metaDataFrame != null)
metaDataFrame.dispose();

close_server();
mainCardLayout.show(mainPanel, "panel1");
}
/*****
//user clicked the twoTier button
void twoTier_button_Clicked(Event event) {
if (textLoginName.getText().length() != 0)
user_name = textLoginName.getText();
else {

```

```

message("Please enter your LogIn Name.");
return;
}
if (txtPassword.getText().length() != 0)
    user_password = txtPassword.getText();
else {
    message("Please enter your Password.");
    return;
}
if (txtDB.getSelectedItemId().length() != 0)
    DB = txtDB.getSelectedItem();
else {
    message("Please enter the Oracle Instance (database name).");
    return;
}
twoTier_connect();
showTables();
}
/*****
// disable_buttons()
void disable_buttons() {
    threeTier_button.setEnabled(false);
    logoff_button.setEnabled(false);
}
/*****/

// message()
void message(String text) {
    current_status_bar = text;
    txtStatus.setText(current_status_bar);
}

// handle_error();
void handle_error() {
    try { close_server(); }
    catch (IOException e) { message("Exception caught! Could not close
server!!!"); }
    disable_buttons();
}
/*****/

// close_server()
void close_server() {
    message("Attempting to close server "+server_URL+"...");

```

```

    try {
        conn.close();
        good_connection = false;
        message("Server closed successfully!");
    }
    catch (IOException e) {
        message("Exception caught! Could not close server!");
        disable_buttons();
    }
}
/*****/
// threeTier_connect()
void threeTier_connect() {
    message("Attempting to open server "+server_URL+"...");

    try {
        t3 = new T3Client("t3://dragonbreath.nosc.mil:7001");
        t3.connect();
        System.out.println("t3 client connected");

        java.util.Properties dbprops = new Properties();
        dbprops.put("user", user_name);
        dbprops.put("password", user_password);
        dbprops.put("server", DB);

        java.util.Properties t3props = new Properties();
        t3props.put("weblogic.t3.dbprops", dbprops);
        t3props.put("weblogic.t3", t3);
        t3props.put("weblogic.t3.driverClassName",
            "weblogic.jdbc.oci.Driver");
        t3props.put("weblogic.t3.driverURL", "jdbc:weblogic:oracle");

        drv = new weblogic.jdbc.t3.Driver();
        conn = drv.connect("jdbc:weblogic:t3", t3props);
        if (conn == null) {
            System.out.println("oracle connection null");
        }
        good_connection = true;

        dma = conn.getMetaData();
        System.out.println("Connected to " + dma.getURL());
    }
}

```

```

dma.getDriverName();
System.out.println("Driver " +
dma.getDriverVersion();
System.out.println("Version " +
dma.getCatalogTerm();
System.out.println("CatalogTerm " +
dma.getSchemaTerm();
System.out.println("SchemaTerm " +
System.out.println("");
message("Server opened successfully!");
logoff_button.setDisabled(true);

} catch (Exception e) {
    message("Exception: " + e);
}
}
}
/*****
// twoTier_connect()
void twoTier_connect() {

    message("Attempting to open server "+server_URL+" ...");

    try {

        java.util.Properties dbprops = new Properties();
        dbprops.put("user", user_name);
        dbprops.put("password", user_password);
        dbprops.put("server", "rqmt");

        Class.forName("weblogic.jdbc.oci.Driver");
        conn = DriverManager.getConnection("jdbc:weblogic:oracle",

        dbprops);

        if (conn == null) {
            System.out.println("oracle connection null");
        }
        good_connection = true;

        dma = conn.getMetaData();

        System.out.println("Connected to " + dma.getURL());

```

```

dma.getDriverName();
System.out.println("Driver " +
dma.getDriverVersion();
System.out.println("Version " +
dma.getCatalogTerm();
System.out.println("CatalogTerm " +
dma.getSchemaTerm();
System.out.println("");
message("Server opened successfully!");
threeTier_button.setDisabled(false);
logoff_button.setDisabled(true);

} catch (Exception e) {
    message("Exception: " + e);
}
}
}
/*****
// odbc_connect()
void odbc_connect() {

    message("Attempting to open server "+server_URL+" ...");

    try {

        java.util.Properties dbprops = new Properties();
        dbprops.put("user", user_name);
        dbprops.put("password", user_password);

        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        conn = DriverManager.getConnection("jdbc:odbc:rqmt",

        dbprops);

        if (conn == null) {
            System.out.println("oracle connection null");
        }
        good_connection = true;

        dma = conn.getMetaData();

        System.out.println("Connected to " + dma.getURL());
        System.out.println("Driver " +
dma.getDriverName());

```

```

dma.getDriverVersion());
System.out.println("Version " +
dma.getCatalogTerm());
System.out.println("CatalogTerm " +
dma.getSchemaTerm());
System.out.println("SchemaTerm " +
System.out.println("");
message("Server opened successfully!");
threeTier_button.setEnabled(false);
logout_button.setEnabled(true);
}
catch (Exception e) {
    message("Exception: " + e);
}
}
}
/*****
/

```

```

//{{DECLARE_CONTROLS for panel1

```

```

CardLayout mainCardLayout;
java.awt.Panel mainPanel;
java.awt.Panel panel1;
java.awt.Panel panel2;

java.awt.TextField textPassword;
java.awt.TextField textLoginName;
java.awt.Label labelPassword;
java.awt.Label labelName;
java.awt.Button threeTier_button;
java.awt.Button twoTier_button;
java.awt.Button jdbc_button;

java.awt.Button logout_button;
java.awt.Button schema_button;
java.awt.Button browse_button;
java.awt.Button edit_button;
java.awt.Button metadata_button;
java.awt.TextArea textStatus;
java.awt.Choice textT3;
java.awt.Label labelT3;
java.awt.Choice textDB;
java.awt.Label labelInstance;

```

```

//{{DECLARE_CONTROLS for panel2
java.awt.List m;
java.awt.Button cb;
java.awt.Label label1;
java.awt.TextField tableNameLabel;
java.awt.Label tableListLabel;
//{{DECLARE_CONTROLS for Frame
BrowseFrame dbFrame;
SchemaFrame schemaPad;
MetaDataFrame metaDataFrame;

}
/* End: MainFrame.java */

```

```

/* File: BrowseFrame.java */

package DbJava;
/*****
import java.awt.*;
import java.sql.*;
*****/

public class BrowseFrame extends Frame {
    ResultSet rs;
    Connection conn1;
    String tableName;
    ResultSet rs1; // result set for all rows of this tables
    String rowid;
    int rowcount;
    int currentRow;

    /*****
    public void destroy() {
        try {
            rs1.close();
        } catch (Exception e) {
            System.out.println("Exception: " + e);
        }
    }

    /*****
    public BrowseFrame() {
        //{{{INIT_CONTROLS
        int buttonWidth = 43;
        int buttonHeight = 43;
        int frameHeight = 400;
        int frameWidth = 1000;
        int scrollPanelHeight = 1000;
        int scrollPanelWidth = 1200;
        int toolbarHeight = 60;
        int scrollbarThickness = 20;

        setLayout(null);
        addNotify();
        setTitle("Table Browser");
        setSize(frameWidth, frameHeight);
        setResizable(false);
    }
    *****/

```

```

        toolbar = new Panel();
        toolbar.setBackground(Color.cyan);
        toolbar.setBounds(
            getInsets().left,
            getInsets().top,
            getBounds().width,
            toolbarHeight);
        add(toolbar);

        panel1 = new Panel();

        panel1.setLayout(null);
        panel1.setBounds(
            getInsets().left,
            getInsets().top+toolbarHeight,
            getBounds().width-scrollbarThickness -
            getInsets().right-getInsets().left,
            getBounds().height-toolbarHeight -
            (getInsets().bottom+getInsets().top)-
            scrollbarThickness);
        panel1.setBackground(Color.yellow);
        add(panel1);

        scrollPanel = new Panel();
        scrollPanel.setLayout(null);
        scrollPanel.setBackground(Color.white);
        scrollPanel.setBounds(0,0,scrollPanelWidth,scrollPanelHeight);
        panel1.add(scrollPanel);

        vScrollbar = new Scrollbar(
            Scrollbar.VERTICAL,
            0,
            panel1.getBounds().height,
            0,
            scrollPanel.getBounds().height);

        vScrollbar.setBounds(
            getBounds().width-scrollbarThickness-getInsets().right,
            getInsets().top+toolbarHeight,
            scrollbarThickness,
            getBounds().height-toolbarHeight-
            (getInsets().bottom+getInsets().top));
    }

```

```

vScrollbar.setUnitIncrement(1);
vScrollbar.setBlockIncrement(1);
add(vScrollbar);

hScrollbar = new Scrollbar(
    Scrollbar.HORIZONTAL,
    0,
    panel1.getBounds().width,
    0,
    scrollPanel.getBounds().width);

hScrollbar.setBounds(
    getInsets().left,
    getBounds().height-scrollbarThickness-
    getBounds().width-scrollbarThickness-getInsets().right-
    getInsets().bottom,
    getInsets().left,
    getInsets().left,
    scrollbarThickness);

hScrollbar.setUnitIncrement(1);
hScrollbar.setBlockIncrement(1);
add(hScrollbar);

fetchButton = new java.awt.Button("fetch");
fetchButton.setBounds(5,5,buttonWidth,buttonHeight);
toolbar.add(fetchButton);
nextButton = new java.awt.Button("next");
nextButton.setBounds(53,5,buttonWidth,buttonHeight);
toolbar.add(nextButton);
//}}

}

public void addFields(ResultSet r, Connection c, String tn) {
    rs = r;
    conn1 = c;
    tableName = tn;
    String aColumnName;
    String aDataType;

    //{{INTL_CONTROLS
    setLayout(null);
    addNotify();

    int x=0;
    int y=0;

    int strLength=256;
    int fieldWidth = 300;

    String tempString;
    int tempInt;

    try {
        while (rs.next()) {
            aColumnName = rs.getString(1);

            JLabel label2 = new java.awt.Label(aColumnName);
            label2.setBounds(6,y+3,strLength,24);
            label2.setBackground(Color.lightGray);
            scrollPanel.add(label2);

            textField1 = new DataRecordTextField("", aColumnName);
            textField1.setBounds(6 + strLength +
                2,y+2,fieldWidth,30);

            textField1.setBackground(Color.white);
            textField1.setEditable(false);
            //textField1.setText(aColumnName); //try for info
            scrollPanel.add(textField1);

            Choice choice1 = new Choice();
            choice1.setBounds(6 + strLength+fieldWidth+
                2,y+2,fieldWidth,30);

            choice1.setBackground(Color.lightGray);
            scrollPanel.add(choice1);
            choice1.addItem("NULL.");

            aDataType = rs.getString(2);
            textField1.setText(aDataType);
            tempInt = rs.getInt(3);
            textField1.setTextLength(tempInt);
            tempInt = rs.getInt(4);
            textField1.setDataPrecision(tempInt);
            tempInt = rs.getInt(5);

```

```

        from " + tn);

        textfield1.setDataScale(tempInt);
        try { //perform query

            Statement stmt1 = conn1.createStatement();
            String sqlStmt = ("Select distinct " + aColumnName + "

            System.out.println("sql: " + sqlStmt);
            stmt1.executeUpdate(sqlStmt);
            ResultSet rs = stmt1.getResultSet();
            while (rs.next()) {
                if (rs.getString(1) != null)

            choice1.addItem(rs.getString(1));
        }

        rs.close();
        stmt1.close();

    } catch (Exception e) {
        System.out.println("Exception occurred running query"
        + e);
    }

    label2 = null;
    textfield1 = null;
    y = y + 30;
    } catch (Exception e) {
        System.out.println("Exception: " + e);
    }

}

/*****
public String getColumnNamesFromFields() {
    Component tempComp;
    String tempString="";
    int n = scrollPanel.getComponentCount();
    for (int i=1; i<n; i++) {
        tempComp = scrollPanel.getComponent(i);
        if (tempComp instanceof JTextField) {
            DataRecordTextField tf = (DataRecordTextField)
            tempComp;

            tempString = tempString + " " + tf.getDataName();
        }

        // need to delete last comma in column name string
        int charCount = tempString.length();
        tempString = tempString.substring(1, charCount);
        return tempString;
    }
    /*****
    public boolean handleEvent(Event event) {
        if (event.id == Event.WINDOW_DESTROY) {
            setVisible(false); // hide the Frame
            dispose(); // free the system resources

            return true;
        }
        else if (event.target == vScrollbar) {
            scrollPanel.setLocation(
                -vScrollbar.getValue());
            //System.out.println("sb: " + vScrollbar.getValue());
        }
        else if (event.target == hScrollbar) {
            scrollPanel.setLocation(
                -hScrollbar.getValue(),
                scrollPanel.getLocation().y);
            //System.out.println("sb: " + hScrollbar.getValue());
        }
        return super.handleEvent(event);
    }
    /*****
    void fetch_button_Clicked(Event event) {
        String tempString="";
        Component tempComp;
        int numComps;

        Cursor cursor1 = getCursor();
        setCursor(new Cursor(Cursor.WAIT_CURSOR));
        try { //perform query
            Statement stmt1 = conn1.createStatement();
            String sqlStmt = (
                "select " + getColumnNamesFromFields() +
                " , ROWID from " + tableName); // NO more Rowid
            System.out.println("sql: " + sqlStmt);

```



```

        return true;
    } else if (label.equalsIgnoreCase("next")) {
        next_button_Clicked(event);
        return true;
    }
}

return super.action(event, arg);
}

/*****
//{{{DECLARE_CONTROLS
java.awt.Panel panel1;
java.awt.Panel scrollPanel;
DataRecordTextField textField1;
TextField rowCountTextField;
java.awt.Panel toolbar;
java.awt.Button fetchButton;
java.awt.Button nextButton;
java.awt.Scrollbar vScrollbar;
java.awt.Scrollbar hScrollbar;
//}}}
}
/* End: BrowseFrame.java */

```

```

/* File: DataRecordTextField.java */

package DbJava;
/***** */
import java.awt.*;
import java.sql.*;
/***** */

class DataRecordTextField extends TextField {
    String dataName;
    String dataType;
    int dataLength;
    int dataPrecision;
    int dataScale;
    boolean updateFlag=false;

    String dataValue;

    /*****/
    public DataRecordTextField(String s, String n) {
        super(s);
        dataName = n;
        updateFlag = false;
    }
    /*****/

    public String getDataName() {
        if (dataType.compareTo("DATE") == 0 && !updateFlag) {
            String s = dataName;
            //s = "TO_CHAR(" + dataName + ", 'YYYY-MM-DD
            HH24:MI:SS')";
            return s;
        }
        else {
            return dataName;
        }
    }
    /*****/
    public String getDataType() {
        return dataType;
    }
    /*****/
    public int getDataLength() {
        return dataLength;
    }
    /*****/
    public int getPrecision() {
        return dataPrecision;
    }
    /*****/
    public int getScale() {
        return dataScale;
    }
    /*****/
    public String getDataValue() {
        return dataValue;
    }
    /*****/

    //
    public void setDataName(String s) {
        dataName = s;
    }
    /*****/
    public void setDataType(String s) {
        dataType = s;
    }
    /*****/
    public void setDataLength(int i) {
        dataLength = i;
    }
    /*****/
    public void setDataPrecision(int i) {
        dataPrecision = i;
    }
    /*****/
    public void setDataScale(int i) {
        dataScale = i;
    }
    /*****/
    public void setUpdateFlag(boolean b) {
        updateFlag = b;
    }
    /*****/
    //
    public boolean validateDataType() {
        if (dataType.compareTo("CHAR") == 0 ||
            dataType.compareTo("VARCHAR2") == 0) {
            if (getLength() > getDataLength()) {

```

```

too many characters");
    System.out.println("Field " + getDataName() + " has
    return false;
    } else if (getLength() == 0) {
        dataValue = "NULL";
    } else {
        dataValue = "" + getLength().trim() + "";
    }
    } else if (dataType.compareTo("NUMBER") == 0) {
        try {
            Integer test = new Integer(getLength().trim());
            dataValue = getLength();
        } catch (NumberFormatException e) {
            System.out.println("Field " + getDataName() + " is not
            of NUMBER type");
            System.out.println("Should be of length " +
            " and scale " + getScale());
            return false;
        }
    } else if (dataType.compareTo("DATE") == 0) {
        try {
            dataValue = "TO_DATE(" + getLength() + ", 'YYYY-
            MON-DD HH24:MI:SS')";
        } catch (NumberFormatException e) {
            System.out.println("date format incorrect: DD-MMM-
            YY");
            return false;
        }
    }
    return true;
}
}
/*****
public boolean validateDataLength() {
    return false;
}
/*****
public boolean validateDataPrecision() {
    return false;
}

```

```

/*****
public boolean validateDataScale() {
    return false;
}
}
/* End: DataRecordTextField.java */

```

```

/* File: EditFrame.java */

package DbJava;
/*****
import java.awt.*;
import java.sql.*;
*****/

public class EditFrame extends JFrame {
    ResultSet rs;
    Connection conn1;
    String tableName;
    ResultSet rs1; // result set for all rows of this tables
    String rowid;
    int rowcount;
    int currentRow;

    /*****
    public void destroy() {
        try {
            rs1.close();
        } catch (Exception e) {
            System.out.println("Exception: " + e);
        }
    }

    /*****
    public EditFrame() {
        //{{{INIT_CONTROLS
        int buttonWidth = 43;
        int buttonHeight = 43;
        int frameHeight = 400;
        int frameWidth = 600;
        int scrollPanelHeight = 1000;
        int scrollPanelWidth = 600;
        int toolbarHeight = 60;
        int scrollbarThickness = 20;

        setLayout(null);
        addNotify();
        setTitle("Table Editor");
        setSize(frameWidth, frameHeight);
        setResizable(false);

        toolbar = new JPanel();
        toolbar.setBackground(Color.cyan);
        toolbar.setBounds(
            getInsets().left,
            getInsets().top,
            getBounds().width,
            toolbarHeight);
        add(toolbar);

        panel1 = new JPanel();

        panel1.setLayout(null);
        panel1.setBounds(
            getInsets().left,
            getInsets().top + toolbarHeight,
            getBounds().width - scrollbarThickness - getInsets().right -
            getInsets().left,
            getBounds().height - toolbarHeight -
            (getInsets().bottom + getInsets().top) - scrollbarThickness);
        panel1.setBackground(Color.yellow);
        add(panel1);

        scrollPanel = new JPanel();
        scrollPanel.setLayout(null);
        scrollPanel.setBackground(Color.white);
        scrollPanel.setBounds(0, 0, scrollPanelWidth, scrollPanelHeight);
        panel1.add(scrollPanel);

        vScrollbar = new Scrollbar(VERTICAL,
            0,
            panel1.getBounds().height,
            0,
            scrollPanel.add(getBounds().height);

        vScrollbar.setBounds(
            getBounds().width - scrollbarThickness - getInsets().right,
            getInsets().top + toolbarHeight,
            scrollbarThickness,
            getBounds().height - toolbarHeight -
            (getInsets().bottom + getInsets().top));

```

```

vScrollbar.setUnitIncrement(1);
vScrollbar.setBlockIncrement(1);
add(vScrollbar);

hScrollbar = new Scrollbar(
    Scrollbar.HORIZONTAL,
    0,
    panel1.getBounds().width,
    0,
    scrollPanel.getBounds().width);

hScrollbar.setBounds(
    insets().left,
    insets().height-scrollbarThickness-
    insets().bottom,
    insets().left,
    insets().width-scrollbarThickness-
    insets().right-scrollbarThickness);

hScrollbar.setUnitIncrement(1);
hScrollbar.setBlockIncrement(1);
add(hScrollbar);

fetchButton = new java.awt.Button("fetch");
fetchButton.setBounds(5,5,buttonWidth,buttonHeight);
toolbar.add(fetchButton);
nextButton = new java.awt.Button("next");
nextButton.setBounds(53,5,buttonWidth,buttonHeight);
toolbar.add(nextButton);
clearButton = new java.awt.Button("new record");
clearButton.setBounds(96,5,buttonWidth,buttonHeight);
toolbar.add(clearButton);
addButton = new java.awt.Button("insert");
addButton.setBounds(137,5,buttonWidth,buttonHeight);
toolbar.add(addButton);
updateButton = new java.awt.Button("update");
updateButton.setBounds(177,5,buttonWidth,buttonHeight);
toolbar.add(updateButton);
deleteButton = new java.awt.Button("delete");
deleteButton.setBounds(217,5,buttonWidth,buttonHeight);
toolbar.add(deleteButton);

//}}
}
/*****
public void addFields(ResultSet r, Connection c, String tn) {
    rs = r;
    tableName = tn;

    //{{INIT_CONTROLS
    setLayout(null);
    addNotify();

    int x=0;
    int y=0;
    int strLength=256;
    int fieldWidth = 300;

    String tempString;
    int tempInt;

    try {
        while (rs.next()) {
            tempString = rs.getString(1);

            Button label2 = new java.awt.Button(tempString);
            label2.setBounds(6,y+3,strLength,24);
            label2.setBackground(Color.lightGray);
            scrollPanel.add(label2);

            textField1 = new DataRecordTextField("", tempString);
            textField1.setBounds(6 + strLength +
                2,y+2,fieldWidth,30);

            textField1.setBackground(Color.white);

            tempString = rs.getString(2);
            textField1.setDataType(tempString);
            tempInt = rs.getInt(3);
            textField1.setMaxLength(tempInt);
            tempInt = rs.getInt(4);
            textField1.setDataPrecision(tempInt);
            tempInt = rs.getInt(5);
            textField1.setDataScale(tempInt);

```

```

scrollPanel.add(textField1);

label2 = null;
textField1 = null;
y = y + 30;
} catch (Exception e) {
    System.out.println("Exception: " + e);
}
}
}
/*****
public boolean action(IEvent event, Object arg) {
    String label;
    String tempString = "";
    Component tempComp;
    int numComps;

    if (event.target instanceof Button) {
        label = (String) arg;
        if (label.equalsIgnoreCase("fetch")) {
            Cursor cursor1 = getCursor();
            setCursor(new Cursor(Cursor.WAIT_CURSOR));
            try { //perform query
                Statement stmt1 = conn1.createStatement();
                String sqlStmt = (
                    "select " + getColumnNamesFromFields() +
                    " , ROWID from " + tableName);
                System.out.println("sql: " + sqlStmt);

                stmt1.executeQuery(sqlStmt);

                rs1 = stmt1.getResultSet();
                if (rs1.next()) {
                    numComps = scrollPanel.getComponentCount();
                    for (int i=0; i<numComps; i++) {
                        tempComp =
                            scrollPanel.getComponent(i);
                        if (tempComp instanceof TextField)
                        {
                            DataRecordTextField tf = (DataRecordTextField) tempComp;
                            String tString;
                            try {
                                tf.setText("");
                                col = rs1.findColumn(tf.getDataName());
int

```

```

scrollPanel.add(textField1);

label2 = null;
textField1 = null;
y = y + 30;
} catch (Exception e) {
    System.out.println("Exception: " + e);
}
}
}
/*****
public String getColumnNamesFromFields() {
    Component tempComp;
    String tempString = "";
    int n = scrollPanel.getComponentCount();
    for (int i=1; i<=n; i++) {
        tempComp = scrollPanel.getComponent(i);
        if (tempComp instanceof TextField) {
            DataRecordTextField tf = (DataRecordTextField)
                tempComp;
            tempString = tempString + " , " + tf.getDataName();
        }
    }
    // need to delete last comma in column name string
    int charCount = tempString.length();
    tempString = tempString.substring(1, charCount);
    return tempString;
}
/*****
public boolean handleIEvent(IEvent event) {
    if (event.id == Event.WINDOW_DESTROY) {
        setVisible(false); // hide the frame
        dispose(); // free the system resources

        return true;
    }
    else if (event.target == vScrollBar) {
        scrollPanel.setLocation(
            scrollPanel.getLocation().x,
            -vScrollBar.getValue());
        //System.out.println("sb: " + vScrollBar.getValue());
    }
    else if (event.target == hScrollBar) {

```



```

if (tempComp instanceof JTextField) {
    DataRecordTextField tf = (DataRecordTextField) tempComp;
    if (tf.validateDataType()) {
        tempString = tempString + tf.getDataValue() + ",";
    } else {
        return true;
    }
}

// need to delete last comma in column name string
int charCount = tempString.length();
tempString = tempString.substring(0, charCount-1);
tempString = tempString + ",";
System.out.println(tempString);
try { //perform query
    Statement stmt1 = conn1.createStatement();
    String sqlStmt = tempString;
    int rowCount = stmt1.executeUpdate(sqlStmt);
    System.out.println("# of rows saved: " + rowCount);
    stmt1.close();
} catch (Exception e) {
    System.out.println("Exception: " + e);
}

return true;
}

return super.action(event, arg);
}

/*****
//{{DECLARE_CONTROLS
java.awt.Panel panel1;
java.awt.Panel scrollPanel;
DataRecordTextField textField1;
TextField rowCountTextField;
java.awt.Panel toolbar;
java.awt.Button fetchButton;

```

```

for (int i=0; i<numComps; i++) {
    tempComp = scrollPanel.getComponent(i);
    if (tempComp instanceof JTextField) {
        DataRecordTextField tf = (DataRecordTextField) tempComp;
        if (tf.validateDataType()) {
            tf.setUpdateFlag(true);
            tempString = tempString + tf.getDataName() + "=" +
                tf.getDataValue() + ",";
        } else {
            tf.setUpdateFlag(false);
            return true;
        }
    }

    // need to delete last comma in column name string
    int charCount = tempString.length();
    tempString = tempString.substring(0, charCount-1);
    tempString = tempString + " where ROWID=" + rowid + " and ";
    System.out.println(tempString);
    try { //perform query
        Statement stmt1 = conn1.createStatement();
        String sqlStmt = tempString;
        int rowCount = stmt1.executeUpdate(sqlStmt);
        System.out.println("# of rows update: " + rowCount);
        stmt1.close();
    } catch (Exception e) {
        System.out.println("Exception: " + e);
    }

    return true;
} else if (label.equalsIgnoreCase("insert")) {
    numComps = scrollPanel.getComponentCount();
    tempString = "insert into " + tableName +
        " (" + getColumnNamesFromFields() + ") values (" +
        for (int i=0; i<numComps; i++) {
            tempComp = scrollPanel.getComponent(i);

```



```
java.awt.Button nextButton;  
java.awt.Button clearButton;  
java.awt.Button updateButton;  
java.awt.Button addButton;  
java.awt.Button deleteButton;  
java.awt.Scrollbar vScrollbar;  
java.awt.Scrollbar hScrollbar;
```

```
    //}}
```

```
}
```

```
/* File: EditFrame.java */
```

```

/* File: MetaDataFrame.java */
package Dbjava;
/*****
import java.awt.*;
import java.sql.*;
import symantec.itools.awt.*;
public class MetaDataFrame extends Frame {
    Connection conn;
    public MetaDataFrame(Connection c) {
        conn = c;
        //{{INIT_CONTROLS
        setLayout(null);
        addNotify();
        resize(insets0.left + insets0.right + 493,insets0.top +
insets0.bottom + 501);
        label1 = new java.awt.Label("Table");
        label1.reshape(insets0.left + 24,insets0.top + 0,48,40);
        label1.setFont(new Font("Dialog", Font.BOLD, 12));
        add(label1);
        tableChoice = new java.awt.Choice();
        add(tableChoice);
        tableChoice.reshape(insets0.left + 72,insets0.top + 12,180,21);
        panel1 = new java.awt.Panel();
        panel1.setLayout(null);
        panel1.reshape(insets0.left + 0,insets0.top + 48,456,100);
        panel1.setBackground(new Color(12632256));
        add(panel1);
        showLabel = new java.awt.Label("Show");
        showLabel.reshape(12,0,108,24);
        showLabel.setFont(new Font("Dialog", Font.BOLD, 12));
        panel1.add(showLabel);
        pkCheckbox = new java.awt.Checkbox("Primary Key");
        pkCheckbox.reshape(12,24,192,36);
        panel1.add(pkCheckbox);
        dtCheckbox = new java.awt.Checkbox("Data Types");
        dtCheckbox.reshape(252,24,192,36);
        panel1.add(dtCheckbox);
        fkCheckbox = new java.awt.Checkbox("Foreign Keys");
        fkCheckbox.reshape(12,72,192,36);
        panel1.add(fkCheckbox);
        dcCheckbox = new java.awt.Checkbox("Data Constraints");
        dcCheckbox.reshape(252,72,192,36);
        panel1.add(dcCheckbox);
}
}
*****/
}

```

```

try { //perform query
    Statement stmt1 = conn.createStatement();
    String sqlStmt = (
        "select
        column_name,data_type,data_length,data_precision,data_scale,nullable " +
        "from cols where table_name = " + tableName + " order by
        nullable");
    System.out.println("sql: " + sqlStmt);
    stmt1.executeQuery(sqlStmt);
    ResultSet rs = stmt1.getResultSet();
    int i=0;
    while (rs.next()) {
        String tempString;
        columnList.add(new TableCell(i,0,rs.getString(1)));
        //if (dcCheckBox.getState())

        columnList.add(new TableCell(i,1,rs.getString(2)));
        //if (dcCheckBox.getState())

        columnList.add(new TableCell(i,3,rs.getString(3)));

        columnList.add(new TableCell(i,2,rs.getString(6)));
        i++;
    }
    rs.close();
} catch (Exception e) {
    System.out.println("Exception occurred running query" + e);
}

}
public boolean handleEvent(Event event) {
    if (event.id == Event.WINDOW_DESTROY) {
        hide(); // hide the frame
        return true;
    }
    else if (event.target == metaDataButton && event.id ==
        Event.ACTION_EVENT) {
        metaDataButton_Clicked(event);
    }
}

```

```

return true;
}

return super.handleEvent(event);
}

//{{DECLARE_CONTROLS
java.awt.Label label1;
java.awt.Choice tableChoice;
java.awt.Panel panel1;
java.awt.Label showLabel;
java.awt.CheckBox pkCheckBox;
java.awt.CheckBox dtCheckBox;
java.awt.CheckBox fkCheckBox;
java.awt.CheckBox dcCheckBox;
java.awt.Button metaDataButton;
java.awt.Panel panel2;
java.awt.Button tableButton;
symantec.tools.awt.MultiList columnList;

}
/* File: MetaDataFrame.java */

```

```

}
public void paint(Graphics g) {
    g.setColor(Color.cyan);
    int x=0;
    int y=0;
    int objWidth = 200;
    int objHeight = 200;

    //g.drawLine(0,0,200,200);
    g.drawRect(x,y,objWidth,objHeight);
    g.setColor(Color.blue);
    g.drawLine(x,y+20,x+objWidth,y+20);
    g.drawString(dt.getTableNames(),x+2,y+20);

    Vector aColumns = dt.getColumns();
    for (int i=0;i< aColumns.size(); i++) {
        DataColumn col = (DataColumn)
            aColumns.elementAt(i);

        g.drawString(col.getColumnName(),x+2,y+40+(i*20));
        g.drawString(" "+col.getDataType()+"",
            x+100,y+40+(i*20));
    }

}

/* I'nd: SchemaCanvas.java */

```

```

/* File: SchemaCanvas.java */

package DbJava;
/*****
import java.awt.*;
import java.util.*;
import java.lang.*;
import java.sql.*;
import DbJava.relatinal.*;
*****/
public class SchemaCanvas extends Canvas {

    Canvas canvas;
    DataTable dt;
    ResultSet rs;

    public void setResultSet(ResultSet r) {
        rs = r;
    }

    public void setDataTable(DataTable d) {
        dt = d;
    }

    public ResultSet getResultSet() {
        return rs;
    }

    public DataTable getDataTable() {
        return dt;
    }

    public SchemaCanvas() {
        canvas = new Canvas();
    }

    //
    public boolean mouseDown(MouseEvent e, int x, int y) {
        //System.out.println("mouseDown:" + x + "," + y);
        return true;
    }

    public boolean mouseDrag(MouseEvent e, int x, int y) {
        //System.out.println("mouseDrag:" + x + "," + y);
        return true;
    }

    public boolean mouseUp(MouseEvent e, int x, int y) {
        //System.out.println("mouseUp:" + x + "," + y);
        setLocation(Math.abs(x),Math.abs(y));
        return true;
    }
}

```

```

/* File: SchemaFrame.java */
package DbJava;
/*****
import java.awt.*;
import java.sql.*;
import java.lang.*;
import java.util.*;
import DbJava.relatinal.*;
*****/

public class SchemaFrame extends Frame {
    Connection conn;

    public void setConnection(Connection c){
        conn = c;
    }

    public SchemaFrame(Connection c) {
        //{{{INIT_CONTROLS
        int mainPanWidth = 400;
        int mainPanHeight = 400;
        int containerPanWidth = 2000;
        int containerPanHeight = 2000;

        conn = c;
        setLayout(new BorderLayout(0,0));
        addNotify();
        setTitle("Schema");

        mainPan = new Panel();
        hbar = new Scrollbar(
            Scrollbar.HORIZONTAL,
            0,
            mainPanWidth,
            0,
            containerPanWidth);
        vbar = new Scrollbar(
            Scrollbar.VERTICAL,
            0,
            mainPanHeight,
            0,
            containerPanHeight);

```

```

        add("Center",mainPan);
        add("East",vbar);
        add("South",hbar);

        mainPan.setLayout(null);
        containerPanel = new Panel();
        mainPan.add(containerPanel);
        mainPan.setBounds(0,0,mainPanWidth,mainPanHeight);
        containerPanel.setBackground(Color.blue);
        containerPanel.setBounds(0,0,
            containerPanWidth,
            containerPanHeight);

        //Create instance of database
        DataSchema aDataSchema = new DataSchema();
        Vector aTables = aDataSchema.getTables();

        //Create instance of tables
        try {
            //dma = conn.getMetaData();
            Statement stmt = conn.createStatement();
            stmt.executeQuery("select tname from tab" +
                " where tabtype = 'TABLE' and tname not
                like 'DEI%'");

            ResultSet rs = stmt.getResultSet();

            while (rs.next()) {
                DataTable aDataTable = new
                DataTable(rs.getString(1));
                aTables.addElement(aDataTable);
                Statement stmt2 = conn.createStatement();
                String sqlString = "select" +
                    "
                    column_name,data_type,data_length,data_precision,data_scale" +
                    "from cols where table_name = " + rs.getString(1) +
                    """;

                stmt2.executeQuery(sqlString);

                System.out.println("stmt2: " + sqlString);
                ResultSet rs2 = stmt2.getResultSet();
                Vector aColumns = aDataTable.getColumns();
                //Create instance of columns
                while (rs2.next()) {

```

```

        DataColumn(rs2.getString(1),
            DataColumn aDataColumn = new
                "CHAR",1,0,false);
        aColumns.addElement(aDataColumn);
    }

    } catch (Exception e) {
        System.out.println("Exception:" + e);
    }

    //Draw Canvas with DataSchema
    Vector oTables = aDataSchema.getTables();
    for (int i=0; i < oTables.size(); i++) {
        DataTable oDataTable = (DataTable)
            oTables.elementAt(i);

        SchemaCanvas can = new SchemaCanvas();
        containerPanel.add(can);
        can.setBackground(Color.white);
        can.setBounds((i*210),10,200,200);

        can.setDataTable(oDataTable);
    }

    //}}

    public boolean handleEvent(Event event) {
        if (event.id == Event.WINDOW_DESTROY) {
            setVisible(false);    // hide the frame
            dispose();    // free the system resources

            return true;
        }
        if (event.target == vbar) {
            containerPanel.setLocation(containerPanel.getLocation().x,

```

```

                0-vbar.getValue());
        }

        else if (event.target == hbar) {
            containerPanel.setLocation(0-hbar.getValue(),
                containerPanel.getLocation().y);
        }

        return super.handleEvent(event);
    }

    //{{DECLARING CONTROLS
    Scrollbar hbar;
    Scrollbar vbar;
    Panel mainPan;
    Panel containerPanel;
    //}}

}

/* End: SchemaFrame.java */

```

```

/* File: DataSchema.java */

package DbJava.relational;
/*****
import java.lang.*;
import java.util.*;
*****/

public class DataSchema {
    String schemaName;
    Vector tables;
    //
    public DataSchema() {
        tables = new Vector();
    }
    public DataSchema(Vector v) {
        tables = v;
    }
    //
    public void setSchemaName(String s) {
        schemaName = s;
    }
    public void setTables(Vector v) {
        tables = v;
    }
    //
    public String getSchemaName() {
        return schemaName;
    }
    public Vector getTables() {
        return tables;
    }
}
/* End: DataSchema.java */

```

```

/* File: DataTable.java */

package DbJava.relational;
/*****
import java.lang.*;
import java.util.*;
*****/

public class DataTable {
    String tableName;
    Vector columns;
    //
    public DataTable() {
        tableName = null;
        columns = new Vector();
    }
    public DataTable(String s) {
        tableName = s;
        columns = new Vector();
    }
    public DataTable(String s, Vector v) {
        tableName = s;
        columns = v;
    }
    //
    public void setTableName(String s) {
        tableName = s;
    }
    public void setColumns(Vector v) {
        columns = v;
    }
    public String getTableName() {
        return tableName;
    }
    //
    public Vector getColumns() {
        return columns;
    }
    public void addColumn(DataColumn dc) {
        columns.addElement(dc);
    }
}

/* End: DataTable.java */

```



```

/* File: DataColumn.java */

package DbJava.relatinal;
/*****
import java.lang.*;
import java.util.*;
*****/

public class DataColumn {
    String columnName;
    String dataType;
    int dataPrecision;
    int dataScale;
    boolean nullable;
    boolean primaryKey;
    boolean foreignKey;

    //
    public DataColumn() {
        columnName = null;
        dataType = null;
        dataPrecision = 0;
        dataScale = 0;
        nullable = false;
        primaryKey = false;
        foreignKey = false;
    }

    public DataColumn(String s1, String s2, int i1, int i2, boolean b) {
        columnName = s1;
        dataType = s2;
        dataPrecision = i1;
        dataScale = i2;
        nullable = b;
    }

    //
    public void setName(String s) {
        columnName = s;
    }

    public void setType(String s) {
        dataType = s;
    }

    public void setPrecision(int i) {
        dataPrecision = i;
    }

    public void setScale(int i) {
        dataScale = i;
    }
}

```

```

    public void setNullable(boolean b) {
        nullable = b;
    }

    public void setPrimaryKey(boolean b) {
        primaryKey = b;
    }

    public void setForeignKey(boolean b) {
        foreignKey = b;
    }

    //
    public String getColumnName() {
        return columnName;
    }

    public String getDataType() {
        return dataType;
    }

    public int getDataPrecision() {
        return dataPrecision;
    }

    public int getDataScale() {
        return dataScale;
    }

    public boolean getNullable() {
        return nullable;
    }

    public boolean isPrimaryKey() {
        return primaryKey;
    }

    public boolean isForeignKey() {
        return foreignKey;
    }

    }
    /* End: DataColumn.java */

```


LIST OF REFERENCES

1. Hamilton, G. & Cattell, R. JDBC: A Java SQL API, Sun Microsystems Inc., 1996.
2. Johnson, J., "Java as an Application Development Language", Object Magazine, SIGS Publication, June 1996, p. 59.
3. Werman, A. "J/SQL: Taking the pain out of JDBC", Java Report, SIGS Publication, March 1997.
4. Werman, A. "Your first JDBC program", Java Report, June 1996.
5. Duergo, D. & Benson, A., "Representing objects in relational databases", Java Report, SIGS Publications, April 1997.
6. Duhl, J., "Integrating Objects with Relational Data: Exploring Mapping Issues", Object Magazine, SIGS Publications, July 1996.
7. Gamma, E. & et al, Design Patterns, Addison-Wesley Publishing Company, 1995.
8. Orfali, R. & Harkey, D., Client/Server Programming with JAVA and CORBA, John Wiley & Sons, Inc., 1997.
9. Mowbray, T. & Zahavi, R. The Essential CORBA, John Wiley & Sons, Inc., 1995.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center 2
8725 John J. Kingman Road, Suite 0944
Fort Belvoir, VA 22060-6218
2. Dudley Knox Library. 2
Naval Postgraduate School
411 Dyer Rd.
Monterey, CA 93943-5101
3. Center for Naval Analysis. 1
4401 Ford Ave.
Alexandria, VA 22302
4. Dr. Ted lewis, Chairman, Code CS/LT. 1
Computer Science Dept.
Naval Postgraduate School
Monterey, CA 93943
5. Chief of Naval Research. 1
800 North Quincy St.
Arlington, VA 22217
6. Dr. Luqi, Code CS/Lq 1
Computer Science Dept.
Naval Postgraduate School
Monterey, CA 93943
7. Dr. Marvin Langston. 1
1225 Jefferson Davis Highway
Crystal Gateway 2 / Suite 1500
Arlington,VA 22202-4311
8. David Hislop 1
U.S. Army Research Office
PO Box 12211
Research Triangle Park, NC 27709-2211
9. Capt. Talbot Manvel. 1
Naval Sea Systems
2531 Jefferson Davis Hwy.
Attn: TMS 378 Capt. Manvel
Arlington,VA 22240-5150
10. CDR Michael McMahon. 1
Naval Sea System Command
2531 Jefferson Davis Hwy.
Arlington, VA 22242-5160
11. Dr. Elizabeth Wald 1
Office Of Naval Research
800 N. Quincy St.
ONR CODE 311
Arlington, VA 22132-5660

12. Dr. Ralph Wachter. 1
Office of Naval Research
800 N. Quincy St.
CODE 311
Arlington, VA 22217-5660

13. Army Research Lab. 1
115 O'Keefe Building
Attn: Mark Kendall
Atlanta, GA 30332-0862

14. National Science Foundation. 1
Attn: Bruce Barnes
Div. Computer & Computation Research
1800 G St. NW
Washington, DC 20550

15. National Science Foundation. 1
Attn: Bill Agresty
4201 Wilson Blvd.
Arlington, VA 22230

16. Hon. John W. Douglas 1
Assistant Secretary of the Navy
(Inferences, Research, Development and Aquisition)
Room E741
1000 Navy Pentagon
Washington, DC 20350-1000

17. Mike DaBose. 1
NCCOSC / NRaD
Code D4525
Building A33 Room 047A
e-mail: dabose@nosc.mil
Phone (619) 553-6095

18. Joseph D. Barrus. 1
Software Engineer
Naval Command Control and Ocean Surveillance Center
Code N318
San Diego, CA 92152
(619) 553-2706

19. Gary Garingo. 1
NRaD Code D451D
53560 Hull Street
San Diego, CA 92152
(619) 553-6094